# **Java Collections**

<http://javahungry.blogspot.com/2015/05/50-java-collections-interview-questions-and-answers.html>

**Q1 What is the difference between ArrayList and Vector ?**

It is one of the frequently asked collection interview question , the main differences are  
Vector is synchronized while ArrayList is not . Vector is slow while ArrayList is fast . Every time when needed, Vector increases the capacity twice of its initial size while ArrayList increases its ArraySize by 50%. find detailed explanation   [ArrayList vs Vector](http://javahungry.blogspot.co.uk/2013/12/difference-between-arraylist-and-vector-in-java-collection-interview-question.html" \t "_blank)  .

**Arraylist vs Vector in Java**  
  
**1.  Synchronization and Thread-Safe**  
**Vector is  synchronized while ArrayList is not synchronized  .** Synchronization and thread safe means at a time only one thread can access the code .In Vector class all the methods are synchronized .Thats why the Vector object is already synchronized when it is created .  
  
**2.  Performance**  
**Vector is slow as it is thread safe . In comparison ArrayList is fast** as it is non synchronized . Thus     in ArrayList two or more threads  can access the code at the same time  , while Vector is limited to one thread at a time.  
  
**3. Automatic Increase in Capacity**  
**A Vector defaults to doubling size of its array .** While when you insert an element into the ArrayList .**it increases** **its Array size by 50%  .**By default ArrayList size is 10 . It checks whether it reaches the last  element then it will create the new array ,copy the new data of last array to new array ,then old array     is garbage collected by the Java Virtual Machine (JVM) .   
  
**4. Set Increment Size**  
**ArrayList does not define the increment size . Vector defines the increment size .**

You can find the following method in Vector Class  
public synchronized void setSize(int i) { //some code  }  
There is no setSize() method or any other method in ArrayList which can manually set the increment size.  
  
**5. Enumerator**  
**Other than Hashtable ,Vector is the only other class which uses both**[**Enumeration and Iterator**](http://javahungry.blogspot.com/2013/06/difference-between-iterator-and-enumeration-collections-java-interview-question-with-example.html)**.While ArrayList can only use Iterator for traversing an ArrayList .**  
  
**6.  Introduction in Java**  
java.util.Vector  class was there in java since the very first version of the java development kit (jdk).  
java.util.ArrayList  was introduced in java version 1.2 , as part of Java Collections framework . In java version 1.2 , Vector class has been refactored to implement the List Inteface .

**Q2 What is the difference between HashMap and Hashtable ?**  
  
It is one of the most popular collections interview question for java developer . Make sure you go through this once before appearing for the interview .  
Main differences between HashMap and Hashtable are :  
  
a. HashMap allows one null key and any number of null values while Hashtable does not allow null keys and null values.  
b. HashMap is not synchronized or thread-safe while Hashtable is synchronized or thread-safe .  
find detailed explanation here

**Difference between HashMap and HashTable / HashMap vs HashTable**    
  
**1. Synchronization or Thread Safe :**  This is the most important difference between two . HashMap is non synchronized and not thread safe.On the other hand, HashTable is thread safe and synchronized.  
When to use HashMap ?  answer is if your application do not require any multi-threading task, in other words hashmap is better for non-threading applications. HashTable should be used in multithreading applications.   
  
**2. Null keys and null values :**  Hashmap allows one null key and any number of null values, while Hashtable do not allow null keys and null values in the HashTable object.  
  
**3. Iterating the values:**  Hashmap object values are iterated by using iterator .HashTable is the only class other than vector which uses enumerator to iterate the values of HashTable object.

**4.  Fail-fast iterator**  : The iterator in Hashmap is fail-fast iterator while the enumerator for Hashtable is not.  
According to [Oracle Docs](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html),  if the Hashtable is structurally modified at any time after the iterator is created in any way except the iterator's own remove method , then the iterator will throw ConcurrentModification Exception.  
Structural modification means adding or removing elements from the Collection object (here hashmap or hashtable) . Thus the enumerations returned by the Hashtable keys and elements methods are not fail fast  
  
**5. Performance :**  Hashmap is much faster and uses less memory than Hashtable as former is unsynchronized . Unsynchronized objects are often much better in performance in compare to synchronized  object like Hashtable in single threaded environment.  
  
**6. Superclass and Legacy :**  Hashtable is a subclass of Dictionary class which is now obsolete in Jdk 1.7 ,so ,it is not used anymore. It is better off externally synchronizing a HashMap or using a ConcurrentMap implementation (e.g ConcurrentHashMap).HashMap is the subclass of the AbstractMap class. Although Hashtable and HashMap has different superclasses but they both are implementations of the *"Map"*  abstract data type.

**Q18 What is the difference between peek(),poll() and remove() method of the Queue interface ?**  
  
Both poll() and remove() method is used to remove head object of the Queue. The main difference lies when the Queue is empty().  
If Queue is empty then poll() method will return null . While in similar case , remove() method will throw NoSuchElementException .  
peek() method retrieves but does not remove the head of the Queue. If queue is empty then peek() method also returns null.

**Q20 What is the difference between Array and ArrayList in Java ?**  
  
This question checks whether student understand the concept of static and dynamic array. Some main differences between Array and ArrayList are :  
a. Array is static in size while ArrayList is dynamic in size.  
b. Array can contain primitive data types while ArrayList can not contain primitive data types.

**Difference between Array and ArrayList in Java with Example**  
  
**1. Resizable :**   Array is static in size that is fixed length data structure, One can not change the length after creating the Array object.  
ArrayList is dynamic in size . Each ArrayList object  has instance variable *capacity*which indicates the size of the ArrayList. As elements are added to an ArrayList its capacity grows automatically.  
  
**2. Performance :** Performance of Array and ArrayList depends on the operation you are performing :  
*resize() opertation :* Automatic resize of ArrayList will slow down the performance as it will use temporary array to copy elements from the old array to new array.  
ArrayList is internally backed by Array during resizing  as it calls the native implemented method System.arrayCopy(src,srcPos,dest,destPos,length) .  
*add() or get() operation :* adding an element or retrieving an element from the array or arraylist object has almost same  performance , as for ArrayList object these operations  run in constant time.  
  
**3. Primitives :**  ArrayList can not contains primitive data types (like int , float , double) it can only contains Object while Array can contain both primitive data types as well as objects.  
One get a misconception that we can store primitives(int,float,double) in ArrayList , but it is not true    
  
Suppose we have ArrayList object ,

ArrayList  arraylistobject = new ArrayList();  
arraylistobject.add(**23**);  // try to add 23 (primitive)

JVM through Autoboxing(converting primitives to equivalent objects internally) ensures that only objects are added to the arraylist object.   
thus , above step internally works like this :

arraylistobject.add( **new Integer(23)**);         
// Converted int primitive to Integer object and added to arraylistobject

**4. Iterating the values :** We can use iterator  to iterate through ArrayList . The iterators returned by the ArrayList class's iterator and listiterator method are [fail-fast](http://javahungry.blogspot.ca/2014/04/fail-fast-iterator-vs-fail-safe-iterator-difference-with-example-in-java.html).  We can use for loop or for each loop to iterate through array .    
  
**5. Type-Safety :**In Java , one can ensure Type Safety through Generics. while Array is a homogeneous data structure , thus it will contain objects of specific class or primitives of specific  data type. In array if one try to store the different data type other than the specified while creating the array object , ArrayStoreException is thrown.  
  
for example :

String temp[] =  new String[2];  // creates a string array of size 2  
temp[0] = new Integer(12); // throws ArrayStoreException, trying to add Integer object in String[]   
  
**6. Length :**Length of the ArrayList is provided by the size() method while Each array object has the length variable which returns the length of the array.  
for example :

Integer arrayobject[] = new Integer[3];  
arraylength= arrayobject.length   ;  //uses arrayobject length variable

ArrayList  arraylistobject = new ArrayList();  
arraylistobject.add(12);   
arraylistobject.size();   //uses arraylistobject size method  
  
**7. Adding elements :** We can insert elements into the arraylist object using the add() method while  in array we insert elements using the assignment operator.  
for example :

Integer addarrayobject[] = new Integer[3];  
addarrayobject[0]= new Integer(8)   ;  //new object is added to the array object

**8. Multi-dimensional :**Array can be multi dimensional , while ArrayList is always single dimensional.  
example of multidimensional array:

Integer addarrayobject[][] = new Integer[3][2];  
addarrayobject[0][0]= new Integer(8)

**Q21 What is the difference between HashSet and TreeSet ?**  
  
Main differences between HashSet and TreeSet are :  
a.  HashSet maintains the inserted elements in random order while TreeSet maintains elements in the sorted order  
b. HashSet can store null object while TreeSet can not store null object.  
find detailed explanation here [TreeSet vs HashSet in Java](http://javahungry.blogspot.co.uk/2014/03/difference-between-hashset-and-treeset-similarities-and-example.html" \t "_blank)  
**Difference between HashSet and TreeSet**   
  
**1. Ordering :** HashSet stores the object in random order . There is no guarantee that the element we  inserted first in the HashSet  will be printed first in the output . For example   

**import** **java.util.HashSet**;

**public** **class** **HashSetExample** {

**public** **static** **void** **main**(String[] args) {

HashSet<String> obj1= **new** HashSet<String>();

obj1.add("Alive");

obj1.add("is");

obj1.add("Awesome");

System.out.println(obj1);

}

}

**OUTPUT :** **[is, Awesome, Alive]**

Elements are sorted according to the natural ordering of its elements in TreeSet. If the objects can not   
be sorted in natural order than use [compareTo() method to sort the elements](http://javahungry.blogspot.com/2013/08/difference-between-comparable-and.html" \t "_blank) of TreeSet object .

**import** **java.util.TreeSet**;

**public** **class** **TreeSetExample** {

**public** **static** **void** **main**(String[] args) {

TreeSet<String> obj1= **new** TreeSet<String>();

obj1.add("Alive");

obj1.add("is");

obj1.add("Awesome");

System.out.println(obj1);

}

}

**OUTPUT : [Alive, Awesome, is]**

**2. Null value :**   HashSet can store null object while TreeSet does not allow null object. If one try to store null object in TreeSet object , it will throw Null Pointer Exception.  
  
**3. Performance :** HashSet take constant time performance for the basic operations like add, remove contains and  size.While TreeSet guarantees log(n) time cost for the basic operations (add,remove,contains).  
  
**4. Speed :** HashSet is much faster than TreeSet,as performance time of HashSet is constant against the log time of TreeSet for most operations (add,remove ,contains and size) . Iteration performance of HashSet mainly depends on the load factor and initial capacity parameters.   
  
**5. Internal implementation :**  As we have already discussed [How hashset internally works in java](http://javahungry.blogspot.com/2013/08/how-sets-are-implemented-internally-in.html)thus, in one line HashSet are internally backed by hashmap. While TreeSet is backed by a  Navigable  TreeMap.

**6. Functionality :**    TreeSet is rich in functionality as compare to HashSet. Functions like pollFirst(),pollLast(),first(),last(),ceiling(),lower() etc. makes TreeSet easier to use than HashSet.

**7. Comparision :** HashSet uses equals() method for comparison in java while TreeSet uses compareTo() method for maintaining ordering .  
  
**To whom priority is given TreeSet comparator or Comparable.compareTo() .**  
  
Suppose there are elements in TreeSet which can be naturally sorted by the TreeSet , but we also added our own sorting method by implementing Comparable interface compareTo() method .  
Then to whom priority is given.  
  
  
Answer to the above question is that the Comparator passed into the TreeSet constructor has been given priority.  
According to [Oracle Java docs](http://docs.oracle.com/javase/6/docs/api/java/util/TreeSet.html#TreeSet%28java.util.Comparator%29)   
  
public TreeSet(Comparator comparator)  
  
Constructs a new, empty tree set, sorted according to the specified comparator.  
  
   *Parameters:*  
     
   *comparator* - the comparator that will be used to order this set. *If null*, the natural ordering of the elements will be used.  
  
**Similarities Between HashSet and TreeSet**  
  
**1. Unique Elements :**   Since HashSet and TreeSet both implements Set interface . Both are allowed to store only unique elements in their objects. Thus there can never be any duplicate elements inside the HashSet and TreeSet objects.  
  
**2.** **Not Thread Safe :** HashSet and TreeSet both are not synchronized or not thread safe.HashSet and TreeSet, both implementations are not synchronized. If multiple threads access a hash set/ tree set concurrently, and at least one of the threads modifies the set, it must be synchronized externally.  
  
**3. Clone() method copy technique:**  Both HashSet and TreeSet uses shallow copy technique to create a clone of  their objects .  
  
**4. Fail-fast Iterators :**  The iterators returned by this class's  method are fail-fast: if the set is modified at any time after the iterator is  created, in any way except through the iterator's own  remove method, the iterator will throw a  ConcurrentModificationException.  Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at   an undetermined time in the future.  
  
  
**When to prefer TreeSet over HashSet**  
  
1.  Sorted unique elements are required instead of unique elements.The sorted list given by TreeSet is always in ascending order.  
  
2.   TreeSet has greater locality than HashSet.  
  
If two entries  are near by in the order , then TreeSet places them near each other in data structure and hence in memory, while HashSet spreads the entries all over memory  regardless of the keys they are associated to.   
       
As we know Data reads from the hard drive takes much more latency time than data read from the cache or memory. In case data needs to be read from hard drive than prefer TreeSet as it has greater locality than HashSet.  
  
3. TreeSet uses Red- Black tree algorithm underneath to sort out the elements. When one need to perform read/write operations frequently , then TreeSet is a good choice.  
  
Thats it for the difference between HashSet and TreeSet , if you have any doubts then please mention in the comments.

**Q23 What is the difference between HashMap and ConcurrentHashMap ?**  
  
This is also one of the most popular java collections interview question . Make sure this question is in your to do list before appearing for the interview .  
Main differences between HashMap and ConcurrentHashMap are :  
a. HashMap is not synchronized while ConcurrentHashMap is synchronized.  
b. HashMap can have one null key and any number of null values while ConcurrentHashMap does not allow null keys and null values .  
find detailed explanation here [ConcurrentHashMap vs HashMap in Java](http://javahungry.blogspot.co.uk/2014/02/hashmap-vs-concurrenthashmap-java-collections-interview-question.html" \t "_blank)  
  
**Q24 Arrange the following in the ascending order (performance):**  
**HashMap , Hashtable , ConcurrentHashMap and Collections.SynchronizedMap**  
  
Hashtable  <  Collections.SynchronizedMap  <  ConcurrentHashMap  <  HashMap

**Q26 What is the difference between LinkedList and ArrayList in Java ?**  
  
Main differences between LinkedList and ArrayList are :  
a. LinkedList is the doubly linked list implementation of list interface , while , ArrayList is the resizable array implementation of list interface.  
b. LinkedList can be traversed in the reverse direction using descendingIterator() method  provided by the Java Api developers , while , we need to implement our own method to traverse ArrayList in the reverse direction . find the detailed explanation here [ArrayList vs LinkedList in java](http://javahungry.blogspot.co.uk/2015/04/difference-between-arraylist-and-linkedlist-in-java-example.html" \t "_blank).

**Q28 Why Map interface does not extend the Collection interface in Java Collections Framework ?**  
  
One liner answer : **Map interface is not compatible with the Collection interface.**  
Explanation : Since Map requires key as well as value , for example , if we want to add key-value pair then we will use put(Object key , Object value) . So there are two parameters required to add element to the HashMap object  . In Collection interface add(Object o) has only one parameter.   
The other reasons are Map supports valueSet , keySet as well as other appropriate methods which have just different views from the Collection interface.  
  
**Q29 When to use ArrayList and when to use LinkedList in application?**  
ArrayList has constant time search operation O(1) .Hence, ArrayList is preferred when there are more get() or search operation .  
  
Insertion , Deletion operations take constant time O(1) for LinkedList. Hence, LinkedList is preferred when there are more insertions or deletions involved in the application.  
  
**Q32 What is CopyOnWriteArrayList ?  How it is different from  ArrayList in Java?**  
[CopyOnWriteArrayList](https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) is a thread safe variant of ArrayList   in which all mutative operations like add , set are implemented by creating a fresh copy of the underlying array.  
It guaranteed not to throw ConcurrentModificationException.  
It permits all elements including null. It is introduced in jdk 1.5 .

**Q35 What is BlockingQueue in Java Collections Framework?**  
[BlockingQueue](https://docs.oracle.com/javase/6/docs/api/java/util/concurrent/BlockingQueue.html) implements the java.util.Queue interface . BlockingQueue supports  operations that wait for the queue to become non-empty when retrieving an element , and wait  for space to become available in the queue when storing an element .  
It does not accept null elements.  
Blocking queues are primarily designed for the producer-consumer problems.  
BlockingQueue implementations are thread-safe and can also be used in inter-thread communications.  
This concurrent Collection class was added in jdk 1.5  
  
**Q38 What is the difference between Fail- fast iterator and Fail-safe iterator ?**This is one  of the most popular interview question for the higher experienced java developers .  
Main differences between Fail-fast and Fail-safe iterators are :  
a. Fail-fast throw ConcurrentModificationException while Fail-safe does not.  
b. Fail-fast does not clone the original collection list of objects while Fail-safe creates a copy of the original collection list of objects.  
The difference is explained in detail here [fail-safe vs fail-fast iterator in java](http://javahungry.blogspot.co.uk/2014/04/fail-fast-iterator-vs-fail-safe-iterator-difference-with-example-in-java.html).  
  
**Q40 How do you use a custom object as key in Collection  classes like HashMap ?**  
  
If one is using the custom object as key then one needs to override equals() and hashCode() method  
and one also need to fulfill the contract.  
If you want to store the custom object in the SortedCollections like SortedMap then one needs to make sure that equals() method is consistent to the compareTo() method. If inconsistent , then collection will not follow their contracts ,that is , Sets may allow duplicate elements.

**Q41 What is hash-collision in Hashtable ? How it was handled in Java?**  
  
In Hashtable , if two different keys have the same hash value then it lead to hash -collision. A bucket of type linkedlist used to hold the different keys of same hash value.  
  
**Q42 Explain the importance of hashCode() and equals() method ? Explain the contract also ?**  
HashMap object uses Key object hashCode() method and equals() method to find out the index to put the key-value pair. If we want to get value from the HashMap same both methods are used . Somehow, if both methods are not implemented correctly , it will result in two keys producing the same hashCode() and equals() output. The problem will arise that HashMap will treat both output same instead of different and overwrite the most recent key-value pair with the previous key-value pair.  
Similarly all the collection classes that does not allow the duplicate values use hashCode() and equals() method to find the duplicate elements.So it is very important to implement them correctly.  
  
**Contract of hashCode() and equals() method**  
a.If  object1.equals(object2) , then  object1.hashCode() == object2.hashCode() should always be true.  
  
b. If object1.hashCode() == object2.hashCode() is true does not guarantee object1.equals(object2)  
 **Q43 What is EnumSet in Java ?**  
[EnumSet](http://docs.oracle.com/javase/7/docs/api/java/util/EnumSet.html)  is a specialized Set implementation for use with enum types. All of the elements in an enum set must come from a single enum type that is specified explicitly  or implicitly , when the set is created.  
The iterator never throws ConcurrentModificationException and is weakly consistent.  
Advantage over HashSet:  
All basic operations of EnumSet execute in constant time . It is most likely to be much faster than HashSet counterparts.  
It is a part of Java Collections Framework since jdk 1.5.  
  
**Q44 What are concurrentCollectionClasses?**   
In jdk1.5 , Java Api developers had introduced new package called java.util.concurrent that have thread-safe collection classes as they allow collections to be modified while iterating . The iterator is fail-safe that is it will not throw ConcurrentModificationException.  
Some examples of concurrentCollectionClasses are :  
a. CopyOnWriteArrayList  
b. ConcurrentHashMap  
  
**Q45 How do you convert a given Collection to SynchronizedCollection ?**  
One line code :    Collections.synchronizedCollection(Collection collectionObj) will convert a given collection to synchronized collection.  
  
**Q46  What is IdentityHashMap ?**  
 **IdentityHashMap**  
[IdentityHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/IdentityHashMap.html) is a class present in java.util package. It implements the Map interface with a hash table , using [reference equality instead of object equality](http://javahungry.blogspot.co.uk/2013/06/difference-between-equals-and-double-equals-method-with-example-java-collections-interview-question.html) when comparing keys and values.In other words , in IdentityHashMap two keys k1 and k2 are considered equal if only if (k1==k2).  
IdentityHashMap is not synchronized.  
Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.   
  
**Q47 What is  WeakHashMap ?**  
[WeakHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/WeakHashMap.html) is a class present in java.util package similar to IdentityHashMap. It is a Hashtable based implementation of Map interface with weak keys. An entry in WeakHashMap will automatically be removed when its key is no longer in ordinary use. More precisely the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector.  
It permits null keys and null values.  
Like most collection classes this class is not synchronized.A synchronized WeakHashMap may be constructed using the Collections.synchronizedMap() method.  
Iterators returned by the iterator() method are fail-fast , hence , will throw ConcurrentModificationException.

**Q48 How will you make Collections readOnly ?**

We can make the Collection readOnly by using the following lines code:

General : Collections.unmodifiableCollection(Collection c)  
Collections.unmodifiableMap(Map m)

Collections.unmodifiableList(List l)

Collections.unmodifiableSet(Set s)

**Q49  What is UnsupportedOperationException?**  
This exception is thrown to indicate that the requested operation is not supported.  
Example of UnsupportedOperationException:  
In other words, if you call add() or remove() method on the readOnly collection . We know readOnly collection can not be modified . Hence , UnsupportedOperationException will be thrown.  
 **Q50 Suppose there is an Employee class. We add Employee class objects to the ArrayList. Mention the steps need to be taken , if I want to sort the objects in ArrayList using the employeeId attribute present  in Employee class.**  
a. Implement the Comparable interface for the Employee class and now to compare the objects by employeeId we will override the emp1.compareTo(emp2)  
b. We will now call Collections class sort method and pass the list as argument , that is ,  
     Collections.sort(empList)    
If you want to add more java collections interview questions  and answers or in case you have any doubts related to the Java Collections framework , then please mention in the comments.

**Threaading**

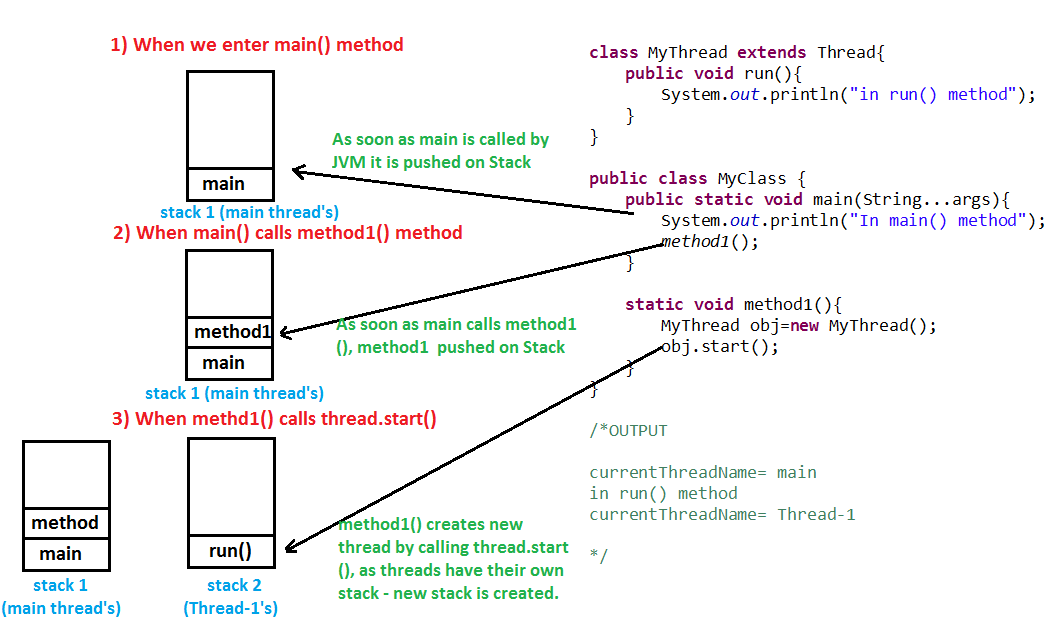
**-----------------**  
[Differences between implementing Runnable interface and extending Thread class](http://www.javamadesoeasy.com/2015/03/differences-between-implementing.html) -

1. Multiple inheritance in not allowed in java : When we [implement Runnable](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) interface we can extend another class as well, but if we extend Thread class we cannot extend any other class because java does not allow multiple inheritance. So, same work is done by implementing Runnable and [extending Thread](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) but in case of implementing Runnable we are still left with option of extending some other class. So, it’s better to implement Runnable.
2. [Thread safety](http://www.javamadesoeasy.com/2015/03/guidelines-to-thread-safe-code-most.html) : When we implement Runnable interface, same object is shared amongst multiple threads, but when we extend Thread class each and every thread gets associated with new object.
3. Inheritance (Implementing Runnable is lightweight operation) : When we extend Thread unnecessary all Thread class features are inherited, but when we implement Runnable interface no extra feature are inherited, as Runnable only consists only of one abstract method i.e. run() method. So, implementing Runnable is lightweight operation.
4. Coding to interface : Even java recommends coding to interface. So, we must implement Runnable rather than extending thread. Also, Thread class implements Runnable interface.
5. Don’t extend unless you wanna modify fundamental behaviour of class, Runnable interface has only one abstract method i.e. run()  : We must [extend Thread](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) only when you are looking to modify run() and other methods as well. If you are simply looking to modify only the run() method [implementing Runnable](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) is the best option (Runnable interface has only one abstract method i.e. run() ). We must not extend Thread class unless we're looking to modify fundamental behaviour of Thread class.
6. Flexibility in code when we implement Runnable : When we extend Thread first a fall all thread features are inherited and our class becomes direct subclass of Thread , so whatever action we are doing is in Thread class. But, when we implement Runnable we create a new thread and pass runnable object as parameter,we could pass runnable object to executorService & much more. So, we have more options when we implement Runnable and our code becomes more flexible.
7. ExecutorService : If we implement Runnable, we can start multiple thread created on runnable object  with ExecutorService (because we can start Runnable object with new threads), but not in the case when we extend Thread (because thread can be started only once).

**Difference between Sleep and Wait method in Java**  
 **1.  Class  belongs :**The wait() method belongs to java.lang.Object class, thus can be called on any Object. The sleep() method belongs to java.lang.Thread class, thus can be called on Threads.  
  
**2. Context :**The wait() method can only be called from Synchronized context i.e. using synchronized block or synchronized method. The sleep() method can be called from any context.  
  
**3. Locking :**The wait() method releases the lock on an object and gives others chance to execute. The sleep() method does not releases the lock of an object for specified time or until interrupt.  
  
**4. Wake up condition :**A waiting thread can be awake by notify() or notifyAll() method. A sleeping can be awaked by interrupt or time expires.  
  
**5. Execution :**Each object has each wait() method for inter-communication between threads. The sleep() method is static method belonging to Thread class. There is a common mistake to write t.sleep(1000) because sleep() is a class method and will pause the current running thread not t.

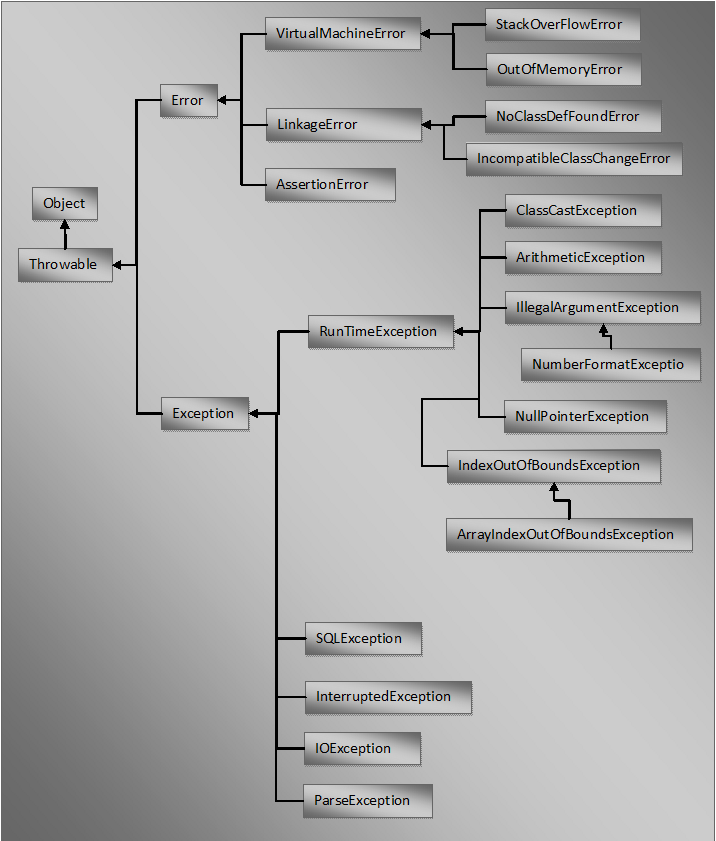
**Does Thread implements their own Stack, if yes how? (Important)**

Answer.  Yes, [Threads have their own stack](http://www.javamadesoeasy.com/2015/03/threads-implement-their-own-stack.html). This is very interesting question, where interviewer tends to check your basic knowledge about how [threads internally maintains their own stacks](http://www.javamadesoeasy.com/2015/03/threads-implement-their-own-stack.html). I’ll be explaining you the concept by diagram.



**Exceptions**

**----------------**



### New features in Java SE 8

* Lambda Expressions
* Pipelines and Streams
* Date and Time API
* Default Methods
* Type Annotations
* Nashhorn JavaScript Engine
* Concurrent Accumulators
* Parallel operations
* PermGen Error Removed
* TLS SNI

## Java Version SE 7

Code named Dolphin and released on July 28, 2011.  


### New features in Java SE 7

* Strings in switch Statement
* Type Inference for Generic Instance Creation
* Multiple Exception Handling
* Support for Dynamic Languages
* Try with Resources
* Java nio Package
* Binary Literals, underscore in literals
* Diamond Syntax
* Automatic null Handling

## Java Version SE 6

Code named Mustang and released on December 11, 2006.



### New features in Java SE 6

* Scripting Language Support
* JDBC 4.0 API
* Java Compiler API
* Pluggable Annotations
* Native PKI, Java GSS, Kerberos and LDAP support.
* Integrated Web Services.
* Lot more enhancements.

## J2SE Version 5.0

Code named Tiger and released on September 30, 2004.



### New features in J2SE 5.0

* Generics
* Enhanced for Loop
* Autoboxing/Unboxing
* Typesafe Enums
* Varargs
* Static Import
* Metadata (Annotations)
* Instrumentation

## J2SE Version 1.4

Code named Merlin and released on February 6, 2002 (first release under JCP).



### New features in J2SE 1.4

* XML Processing
* Java Print Service
* Logging API
* Java Web Start
* JDBC 3.0 API
* Assertions
* Preferences API
* Chained Exception
* IPv6 Support
* Regular Expressions
* Image I/O API

## J2SE Version 1.3

Code named Kestrel and released on May 8, 2000.



### New features in J2SE 1.3

* Java Sound
* Jar Indexing
* A huge list of enhancements in almost all the java area.

## J2SE Version 1.2

Code named Playground and released on December 8, 1998.

### New features in J2SE 1.2

* Collections framework.
* Java String memory map for constants.
* Just In Time (JIT) compiler.
* Jar Signer for signing Java ARchive (JAR) files.
* Policy Tool for granting access to system resources.
* Java Foundation Classes (JFC) which consists of Swing 1.0, Drag and Drop, and Java 2D class libraries.
* Java Plug-in
* Scrollable result sets, BLOB, CLOB, batch update, user-defined types in JDBC.
* Audio support in Applets.

## JDK Version 1.1

## Released on February 19, 1997

### New features in JDK 1.1

* JDBC (Java Database Connectivity)
* Inner Classes
* Java Beans
* RMI (Remote Method Invocation)
* Reflection (introspection only)

**Java is not a pure Object oriented language, but so called a "Hybrid" language.**  
For any language to be pure object oriented it must follow these 6 points strictly...  
**1) It must have full support for Encapsulation and Abstraction**  
**2) It must support Inheritance**  
**3) It must support Polymorphism**  
**4) All predefined types must be Objects**  
**5) All user defined types must be Objects**  
**6) Lastly, all operations performed on objects must be only through methods exposed at the objects.**  
  
Now, java supports 1, 2, 3 & 5 but fails to support **4 & 6.**   
In  java we have some predefined types as non-objects (primitive types).  Although we have wrapper classes for the same but a Pure OOL can't have  anything other than Objects strictly.   
And  regarding point 6, In java we can have communicate with objects without  calling their methods for e.g. using arithmetic operators.  
String s1 = "ABC" + "A" ;  
These 2 points stops java from being a pure OOL.

**List**

*A list is an ordered collection of elements.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Add** | **Remove** | **Get** | **Contains** | **Data  Structure** |
| **ArrayList** | O(1) | O(n) | O(1) | O(n) | Array |
| **LinkedList** | O(1) | O(1) | O(n) | O(n) | Linked List |
| **CopyonWriteArrayList** | O(n) | O(n) | O(1) | O(n) | Array |

**Set**

*A collection that contains no duplicate elements.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Add** | **Contains** | **Next** | **Data Structure** |
| **HashSet** | O(1) | O(1) | O(h/n) | Hash Table |
| **LinkedHashSet** | O(1) | O(1) | O(1) | Hash Table + Linked List |
| **EnumSet** | O(1) | O(1) | O(1) | Bit Vector |
| **TreeSet** | O(log n) | O(log n) | O(log n) | Red-black tree |
| **CopyonWriteArraySet** | O(n) | O(n) | O(1) | Array |
| **ConcurrentSkipList** | O(log n) | O(log n) | O(1) | Skip List |

**Queue**

*A collection designed for holding elements prior to processing.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Offer** | **Peak** | **Poll** | **Size** | **Data Structure** |
| **PriorityQueue** | O(log n ) | O(1) | O(log n) | O(1) | Priority Heap |
| **LinkedList** | O(1) | O(1) | O(1) | O(1) | Array |
| **ArrayDequeue** | O(1) | O(1) | O(1) | O(1) | Linked List |
| **ConcurrentLinkedQueue** | O(1) | O(1) | O(1) | O(n) | Linked List |
| **ArrayBlockingQueue** | O(1) | O(1) | O(1) | O(1) | Array |
| **PriorirityBlockingQueue** | O(log n) | O(1) | O(log n) | O(1) | Priority Heap |
| **SynchronousQueue** | O(1) | O(1) | O(1) | O(1) | None! |
| **DelayQueue** | O(log n) | O(1) | O(log n) | O(1) | Priority Heap |
| **LinkedBlockingQueue** | O(1) | O(1) | O(1) | O(1) | Linked List |

**Map**

*An object that maps keys to values. A map cannot duplicate keys; each key can map to at most one value.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Get** | **ContainsKey** | **Next** | **Data Structure** |
| **HashMap** | O(1) | O(1) | O(h / n) | Hash Table |
| **LinkedHashMap** | O(1) | O(1) | O(1) | Hash Table + Linked List |
| **IdentityHashMap** | O(1) | O(1) | O(h / n) | Array |
| **WeakHashMap** | O(1) | O(1) | O(h / n) | Hash Table |
| **EnumMap** | O(1) | O(1) | O(1) | Array |
| **TreeMap** | O(log n) | O(log n) | O(log n) | Red-black tree |
| **ConcurrentHashMap** | O(1) | O(1) | O(h / n) | Hash Tables |
| **ConcurrentSkipListMap** | O(log n) | O(log n) | O(1) | Skip List |

# **Difference between StringTokenizer and String.split()**

StringTokenizer class allows an application to break a string into tokens. In order to break String into tokens you need to create an object of StringTokenizer class and provide a delimiter for splitting string into tokens. You can pass more then one delimiter at a time e. g you can break string into tokens by : and , at same time. In case if you do not provide any delimiter then by default it takes white-space as a delimiter. It is inferior to split() as it does not support to **regular expression**.  
StringTokenizer is a legacy class that is retained for compatibility reasons although its use is discouraged in new code. It is recommended that is you are looking for such functionality use theString#split() method.

The following is one example of the use of the tokenizer and String.split()

For one thing, StringTokenizer returns one substring at a time whereas the split method returns an array of substrings.  
Best to use String's split(...) method. The StringTokenizer class is a legacy class (NOT deprecated)  
  
By default, StringTokenizer uses whitespace to delimit tokens, and if there are two or more delimiter characters  
in a row, it collapses them into one delimiter. So the split() equivalent would be  
  
String[] result = "this is a test".split("\\s+");  
  
  
But if you really want to treat each character as a separate delimiter, you just remove the plus sign again.  
This is, in my opinion, the best reason to choose split(); while there is a way to do single-character delimiters in StringTokenizer, it's kludgy and awkward.  
  
And yes, StringTokenizer is inherently faster,  
  
StringTokenizer can split strings based on single characters, split() takes regular expressions.  
I imagine that might make split() a bit slower when splitting for single characters (can't be arsed to test it  
though), but usually it's more handy.

From Java 6 javadoc:   
"StringTokenizer is a legacy class that is retained for compatibility reasons although its use is discouraged in new code. It is recommended that anyone seeking this functionality use the split method of String or the java.util.regex package instead."   
  
We should so use .split() function instead.   
I had seen some questions regarding the performance of .split()   
function over Tokenizer, so I decided to give it try.   
  
  
As a test data I used and TCP protocol standard (tons of ASCII text)   
split into chunks to form test data sets of size of   
1 KB.   
  
Findings:   
  
Split is slower than StringTokenizer. I observed that   
StringTokenizer runs consistently 50% faster on a test cases consisting   
of 5 runs for 200 strings of sizes ranging from 1 KB to ~2.50 MB long strings. (1000 total runs)   
  
  
Note that StringTokenizer also appears faster because it "picks" tokens   
as you iterate over it, whereas split() tokenizes string into array at the initial invocation.   
  
Details:   
  
Each run shown is an average of 5 runs on the string of the same size split/tokenized on ",":

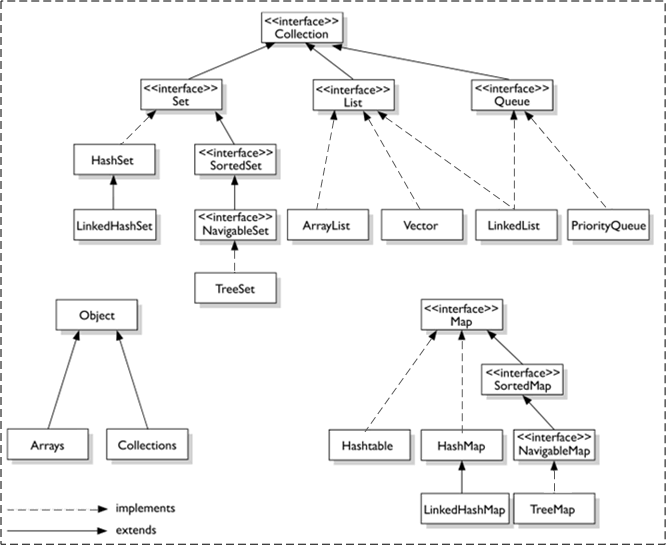
I think the biggest difference is: with a StringTokenizer, the delimiter is just one character long. You supply a list of characters that count as delimiters, but in that list, each character is a single delimiter. With split(), the delimiter is a regular expression, which is something much more powerful (and more complicated to understand). It can be any length. Regular expressions may be harder to understand at first, but when you learn how to use them, they're much more useful.

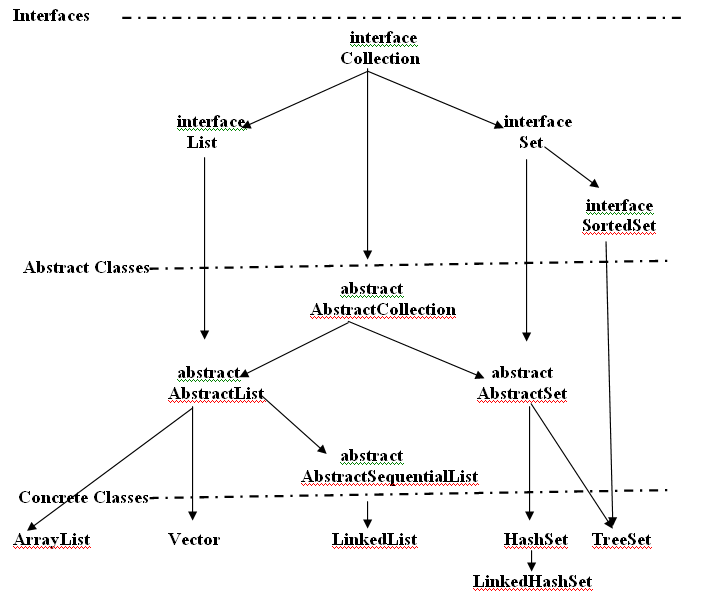
Also, if you need to parse empty tokens, e.g. a comma-separated line like

one,,three,,,six

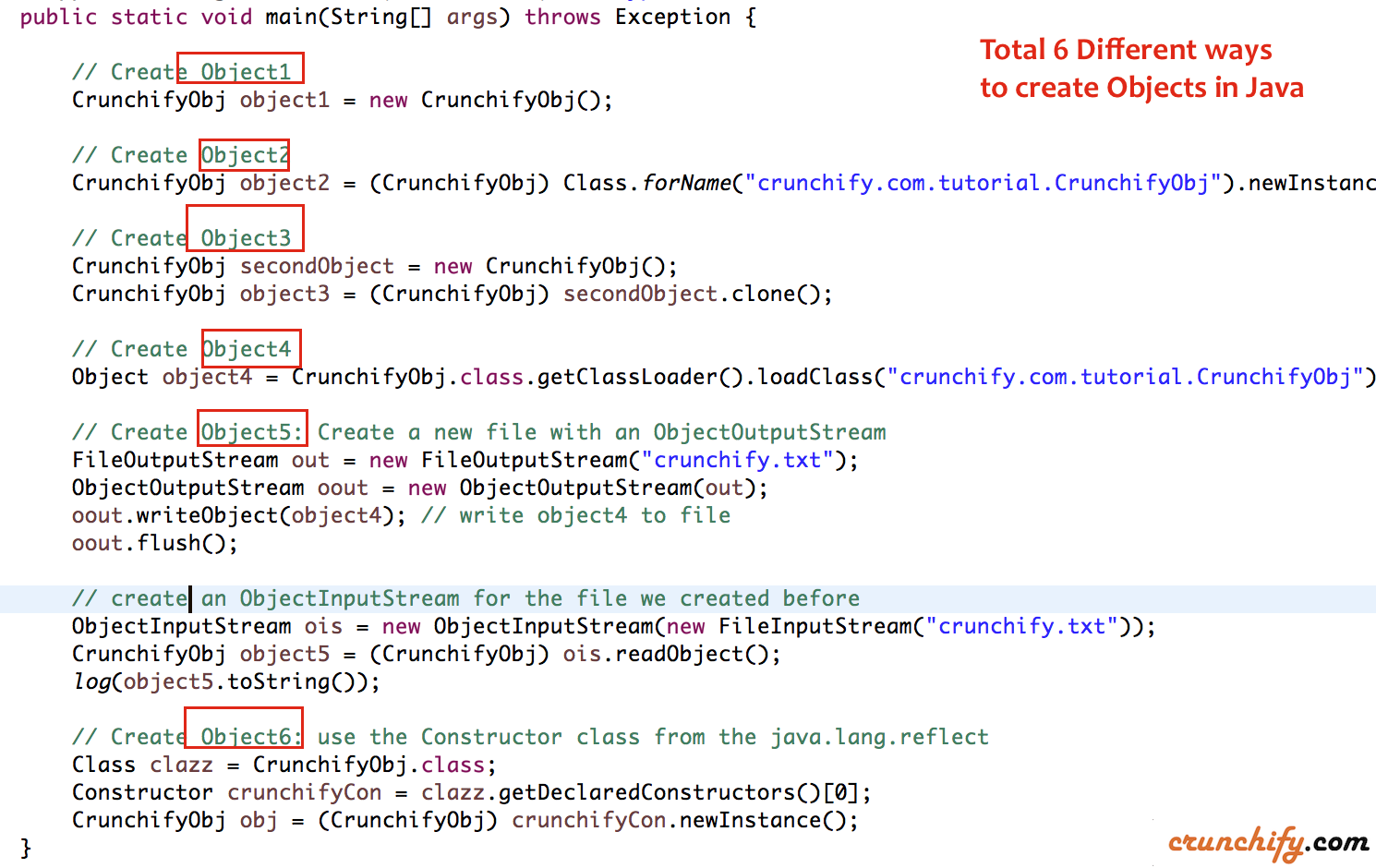
where the field values are "one", "", "three", "", "" and "six" where the three empty strings are indicated by the commas with nothing between them - that's a lot more work with a StringTokenizer. By default it gives you just "one", "three", "six" and skips the empties. You can use a special constructor that takes a boolean to tell theStringTokenizer to return delimiters, but that gets complicated too. I'll skip the details. It's much easier to use split(","), which immediately returns {"one", "", "three", "", "", "six"), exactly right. The short version is: StringTokenizer doesn't handle empty strings well. But split() does.

Some times you want split to behave same as Stringtokenizer then you can use the reqular expressions for acheiving the same effect like spilt(",\*") which takes care of a single , or multiple commas. I you want the delimiter to be , as well as space you can use split("[ ,]") also if you want to trim your strings and has a comma seperated delimiter you can use split(" \*, \*")





# **What are all the Different Ways to Create an Object in Java? Total 6 ways – Complete Tutorial**



As you know, in [Java](http://crunchify.com/category/java-web-development-tutorial/), a class provides the blueprint for objects; you create an object from a class. There are four different ways to create objects in java:

### **Method-1**

Using new keyword. This is the most common way to create an object in java. Almost 99% of objects are created in this way.

Method1: Using new keyword

|  |  |
| --- | --- |
| 1 | CrunchifyObj object = new CrunchifyObj(); |

### **Method-2**

Using Class.forName(). Class.forName() gives you the class object, which is useful for reflection. The [methods](http://crunchify.com/java-static-methods-variables-static-block-and-class-with-example/) that this object has are defined by [Java](http://crunchify.com/category/spring-mvc/), not by the programmer writing the class. They are the same for every class. Calling newInstance() on that gives you an instance of that class (i.e. callingClass.forName("ExampleClass").newInstance() it is equivalent to calling new ExampleClass()), on which you can call the methods that the class defines, access the visible fields etc.

|  |  |
| --- | --- |
| 1 | CrunchifyObj object2 = (CrunchifyObj) Class.forName("crunchify.com.tutorial.CrunchifyObj").newInstance(); |

Class.forName() will always use the ClassLoader of the caller, whereas ClassLoader.loadClass() can specify a different ClassLoader. I believe that Class.forName initializes the loaded class as well, whereas the ClassLoader.loadClass() approach doesn’t do that right away (it’s not initialized until it’s used for the first time).

Another must read:

* [Java: Thread State Introduction with Example](http://crunchify.com/java-thread-state-introduction-with-example/)
* [Simple Java Enum Example](http://crunchify.com/simple-java-enum-example/)

### **Method-3**

Using clone(). The clone() can be used to create a copy of an existing object.

Method3: Using clone()

|  |  |
| --- | --- |
| 1  2 | CrunchifyObj secondObject = new CrunchifyObj();  CrunchifyObj object3 = (CrunchifyObj) secondObject.clone(); |

### **Method-4**

Using newInstance() method

Method4: Using newInstance()

|  |  |
| --- | --- |
| 1 | Object object4 = CrunchifyObj.class.getClassLoader().loadClass("crunchify.com.tutorial.CrunchifyObj").newInstance(); |

### **Method-5**

Using Object Deserialization. Object [Deserialization](http://crunchify.com/how-to-serialize-deserialize-list-of-objects-in-java-java-serialization-example/) is nothing but creating an object from its serialized form.

Method5 - Using object deserialization

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | // Create Object5  // create a new file with an ObjectOutputStream  FileOutputStream out = new FileOutputStream("crunchify.txt");  ObjectOutputStream oout = new ObjectOutputStream(out);    // write something in the file  oout.writeObject(object3);  oout.flush();    // create an ObjectInputStream for the file we created before  ObjectInputStream ois = new ObjectInputStream(new FileInputStream("crunchify.txt"));  CrunchifyObj object5 = (CrunchifyObj) ois.readObject(); |

### **Method-6**

use the Constructor class from the [java.lang.reflect](http://crunchify.com/in-java-how-to-find-list-of-all-class-names-from-inside-jar-file-jar-class-finder-utility/" \t "_blank)

|  |  |
| --- | --- |
| 1  2  3 | Class clazz = CrunchifyObj.class;  Constructor crunchifyCon = clazz.getDeclaredConstructors()[0];  CrunchifyObj obj = (CrunchifyObj) crunchifyCon.newInstance(); |

## Complete Example:

CrunchifyCreateMultipleWaysToCreateObject.java

Java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | package crunchify.com.tutorial;    import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.io.ObjectInputStream;  import java.io.ObjectOutputStream;  import java.io.Serializable;  import java.lang.reflect.Constructor;    /\*\*  \* @author Crunchify.com  \* Total 6 different way to create an Object in Java  \*/    public class CrunchifyObj implements Cloneable, Serializable {    private static final long serialVersionUID = 1L;    public CrunchifyObj() {  log("Hello! CrunchifyObj() just got created..");  }    @Override  protected Object clone() throws CloneNotSupportedException {  return (CrunchifyObj) super.clone();  }    @SuppressWarnings({ "unused", "resource", "rawtypes" })  public static void main(String[] args) throws Exception {    // Create Object1  CrunchifyObj object1 = new CrunchifyObj();    // Create Object2  CrunchifyObj object2 = (CrunchifyObj) Class.forName("crunchify.com.tutorial.CrunchifyObj").newInstance();    // Create Object3  CrunchifyObj secondObject = new CrunchifyObj();  CrunchifyObj object3 = (CrunchifyObj) secondObject.clone();    // Create Object4  Object object4 = CrunchifyObj.class.getClassLoader().loadClass("crunchify.com.tutorial.CrunchifyObj")  .newInstance();    // Create Object5: Create a new file with an ObjectOutputStream  FileOutputStream out = new FileOutputStream("crunchify.txt");  ObjectOutputStream oout = new ObjectOutputStream(out);  oout.writeObject(object4); // write object4 to file  oout.flush();    // create an ObjectInputStream for the file we created before  ObjectInputStream ois = new ObjectInputStream(new FileInputStream("crunchify.txt"));  CrunchifyObj object5 = (CrunchifyObj) ois.readObject();  log(object5.toString());    // Create Object6: use the Constructor class from the java.lang.reflect  Class clazz = CrunchifyObj.class;  Constructor crunchifyCon = clazz.getDeclaredConstructors()[0];  CrunchifyObj obj = (CrunchifyObj) crunchifyCon.newInstance();  }    private static void log(String object) {  System.out.println(object);    }  } |

### **Console Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Hello! CrunchifyObj() just got created..  Hello! CrunchifyObj() just got created..  Hello! CrunchifyObj() just got created..  Hello! CrunchifyObj() just got created..  crunchify.com.tutorial.CrunchifyObj@45ee12a7  Hello! CrunchifyObj() just got created.. |

### Difference between PATH and CLASSPATH in Java

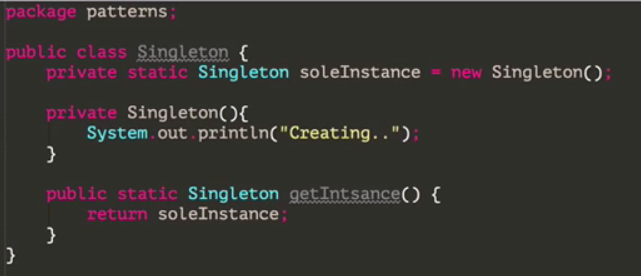
Here are some of the common difference between PATH vs CLASSPATH in Java :

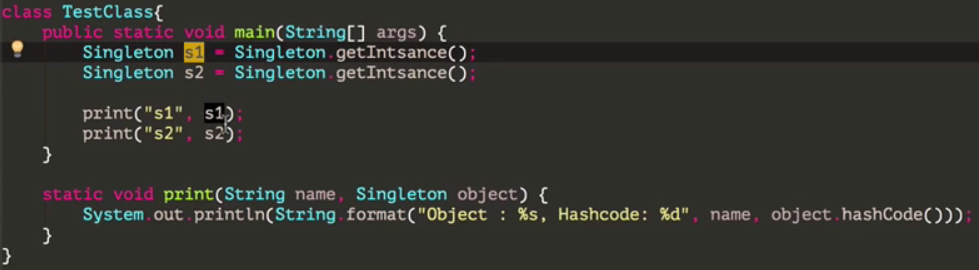
1)Main difference between PATH and CLASSPATH is that  PATH is an environment variable which is used to locate JDK binaries like "java" or "javac" command used to run java program and compile java source file. On the other hand CLASSPATH environment variable is used by System or Application ClassLoader to locate and load compile Java bytecodes stored in .class file.  
  
2) In order to set PATH in Java you need to include JDK\_HOME/bin directory in PATH environment variable while in order to set CLASSPATH in Java you need to include all those directory where you have put either your .class file or JAR file which is required by your Java application.  
  
3) Another significant difference between PATH and CLASSPATH is that PATH can not be overridden by any Java settings but CLASSPATH can be overridden by providing command line option -classpath or -cp to both "java" and "javac" commands or by using Class-Path attribute in Manifest file inside JAR archive.  
  
4) PATH environment variable is used by operating system to find any binary or command typed in shell, this is true for both Windows and Linux environment while CLASSPATH is only used by Java ClassLoaders to load class files.  
  
These were some notable difference between PATH vs CLASSPATH in Java and they are worth remembering to debug and troubleshoot Java related issues.   
  
**How to set PATH and CLASSPATH in Windows and Unix**  
If you are familiar with DOS operating system and how to use command prompt in Windows or shell in Linux setting PATH and CLASSPATH is trivial exercise. Both PATH and CLASSPATH are environment variable and can be set using export in Linux and using set keyword in DOS and Windows as shown below:  
  
Command to set PATH in Windows  
  
set PATH=%PATH%;C:\Program Files\Java\JDK1.6.20\bin  
  
Command to set PATH in UNIX/Linux  
  
export PATH = ${PATH}:/opt/Java/JDK1.6.18/bin  
  
Look at the difference between two commands, in Linux use colon(:) as separator and in Windows use semi-colon(;) as separator.  
  
Command to set CLASSPATH in windows  
  
set CLASSPATH=%CLASSPATH%;C:\Program Files\Java\JDK1.6.20\lib  
  
Command to set CLASSPATH in Unix/Linux  
  
export CLASSPATH= ${CLASSPATH}:/opt/Java/JDK1.6.18/lib  
  
Read more: <http://java67.blogspot.com/2012/08/what-is-path-and-classpath-in-java-difference.html#ixzz4APOfR8kh>

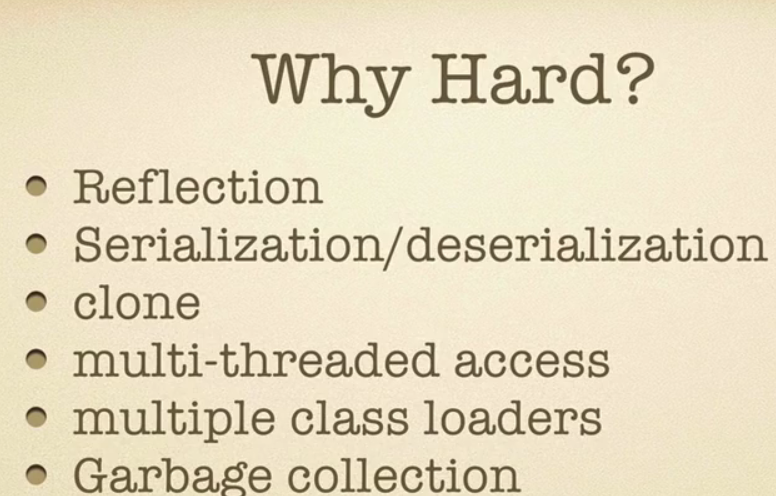
<http://netjs.blogspot.com/2015/08/how-arraylist-works-internally-in-java.html>

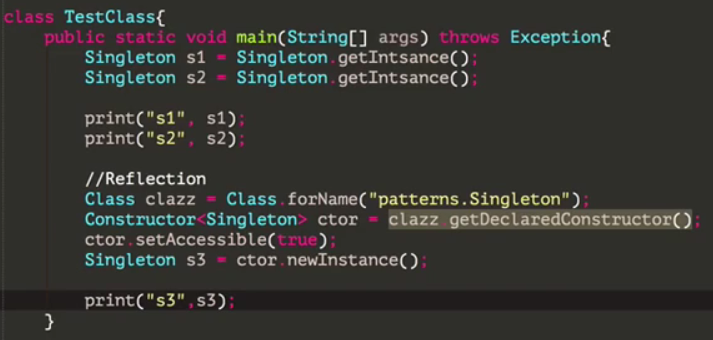
Singletone design pattern

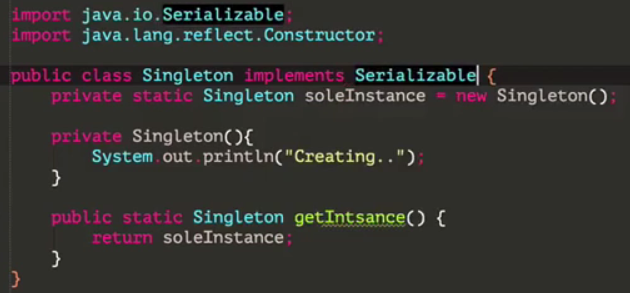
----------------------------------

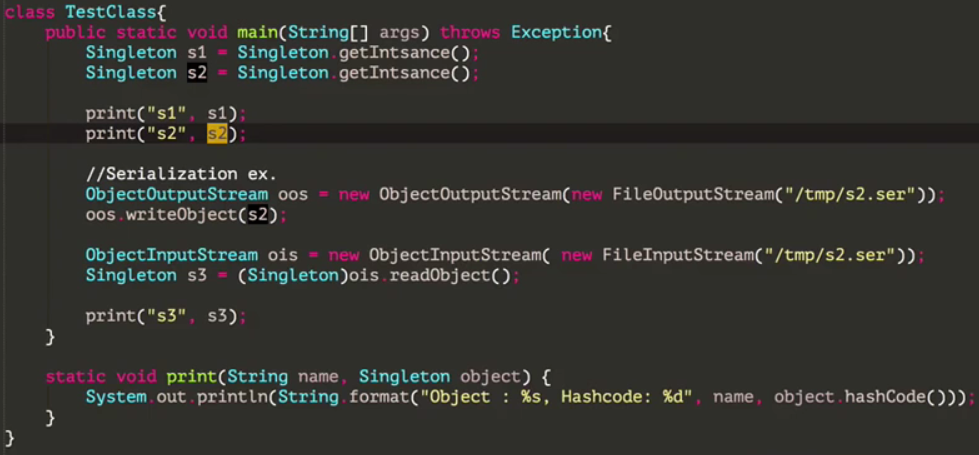


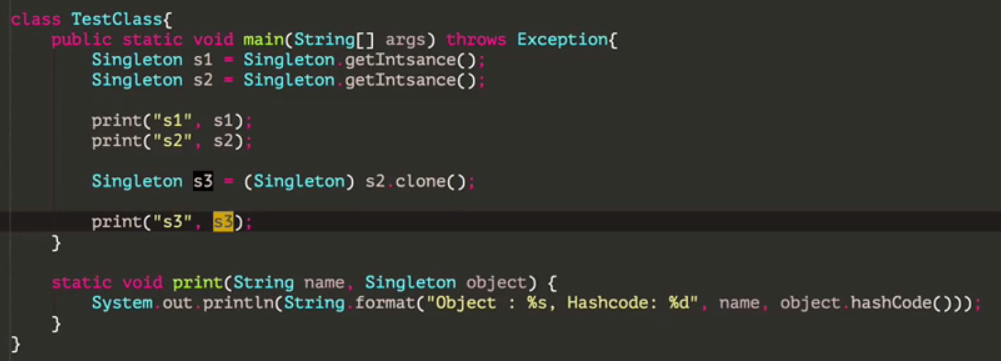


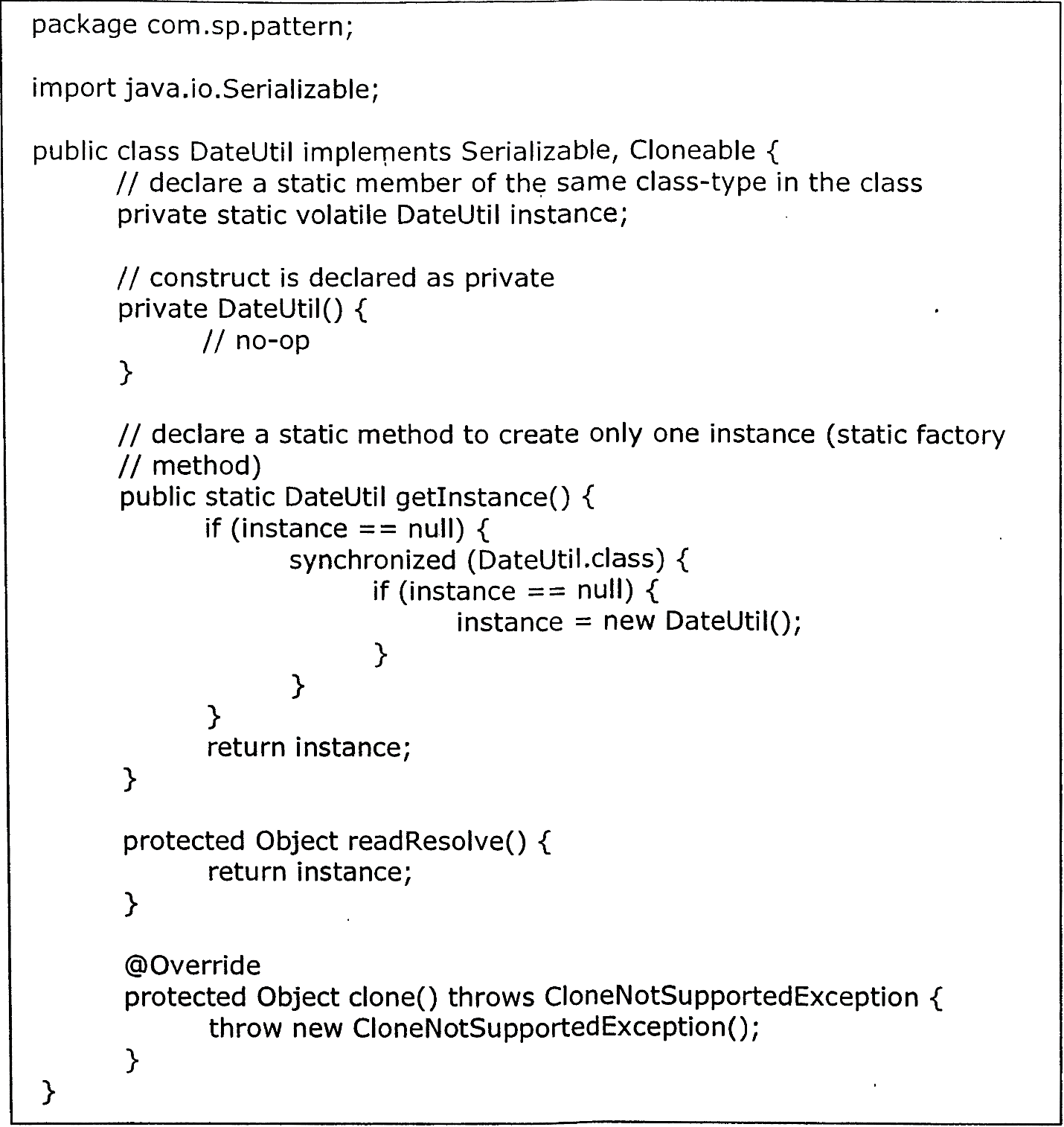


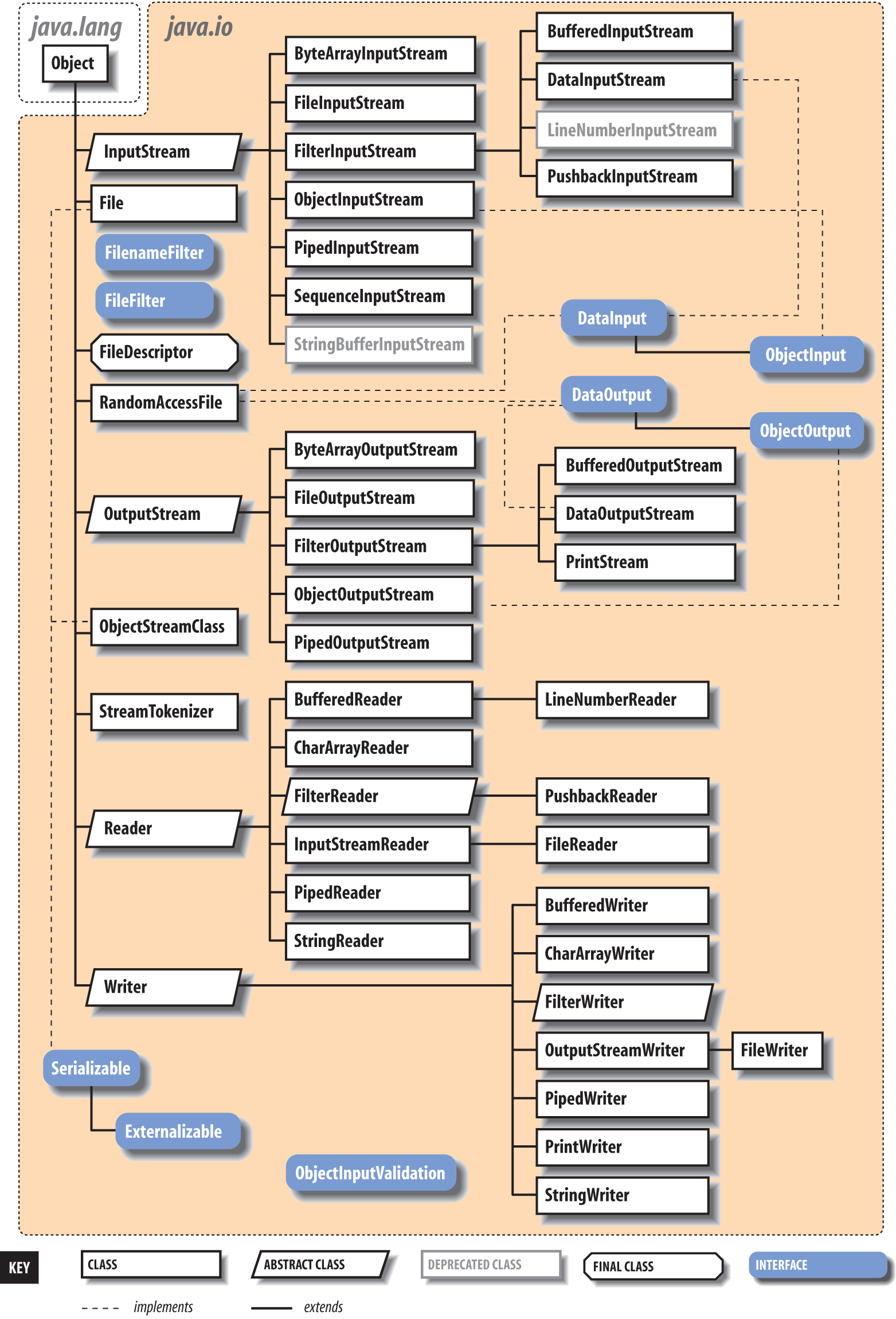












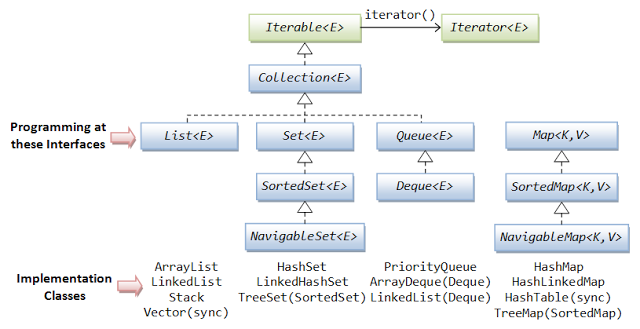
### **Why Map is not a true Collection?**

We saw what are [Collections](http://opensourceforgeeks.blogspot.in/2013/04/collection-in-java.html) in Java. If you recall the diagram from that post Map does not form a part of Collection though it comes under Collection framework. In more technical terms **Map interface does not implement Collection interface.** Knowing this the question that naturally comes in mind is why so? Why can't Map be a part of Collection frame work? Let us see the explanation.

#### **Reason for Map interface not extending Collection interface**

* If you look at the respective data structure you can easily guess why Map is not a part of Collection. Each Collection stores a single value where as a Map stores key-value pair. So methods in Collection interface are incompatible for Map interface.For example in Collection we have **add(Object o).**What would be such implementation in Map. It doesn't make sense to have such a method in Map. Instead we have a **put(key,value)**  method in Map.
* Same argument goes for **addAll**(), **remove**(), **removeAll**() methods. So the main reason is the difference in the way data is stored in Map and Collections.
* Also if you recall Collection interface implemented Iterable interface i.e any interface with .**iterator()** method should return an iterator which must allow us to iterate over the values stored in the Collection. Now what would such method return for a Map? Key iterator or a Value iterator? This does not make sense either.

There are ways in which we can iterate over keys and values stores in a Map and that is how it is a part of Collection framework.Do not get confuse here. Let me repeat this - **Map is a part of Collection framework but it does not implement Collection interface.**We will see in next few posts how Map works internally(that's really an interesting topic) and also how we manipulate data in it.  
  
Just to recall the Collection framework refer to the following diagram

[](http://2.bp.blogspot.com/-uy91eT89jgM/UY-hz2CWHWI/AAAAAAAACdA/doukHuxizAY/s1600/558845_506437529412798_300785732_n.png)

<http://opensourceforgeeks.blogspot.com/2013/05/why-map-is-not-true-collection.html>

**public** **void** creatHashcode() {

            String a="abc";

**int** hash =0;

**for**(**int** i=0;i<a.length();i++){

                  hash=31\*hash+a.charAt(i);

            System.*out*.println("Hash "+hash);

            }

            System.*out*.println("Hash "+a.hashCode() +" abc "+a);